Music Genre Classification using CNN-Hybrid Architectures

Andy Liu
SFU
ala158@sfu.ca

Avital Vetshchaizer
SFU
avetshch@sfu.ca

Henry Yip
SFU
hya59@sfu.ca

Abstract

In this paper we seek to improve the accuracy and runtime performance of the Convolutional Neural Networks (CNN) by introducing multiple modified architectures to achieve better runtime and accuracy. Our proposed models are known as a Convolutional Recurrent Neural Network (CRNN), Duplicated CNN, and Duplicated CRNN. Using the Free Music Archive (FMA) dataset, we investigated different methods of improving the performance of our models by measuring the efficiency, error, and training speed of the final models. As a result, our paper has provided firm evidence that both the duplicated CNN and CRNN have achieved an overall higher test accuracy in a shorter training period, thus proving the applicability and potential usage of these architectures.

1 Introduction

Signal decoding and classification plays a large role in the execution of everyday tasks in the modern world. A large subset of these signals, also known as music, has accompanied humans throughout the eras by evolving to act as symbols and records of the important events at the time. Different styles of music have helped inspire individuals and played large roles in people's self-perception and personal identity. Thus, in this research we attempt to quantify these genres to gain a measurable understanding of their similarities and differences. Our data consists of the spectrum of frequencies and their magnitudes for each song. Once processed, our input data will consist of a spectrogram in an image format. During this project, we experimented with four classification models: CNNs, CRNNs, Duplicated CRNNs (DCRNN), and Duplicated CNNs (DCNN) to attempt to gain a higher confidence rate above the standard CNN. To achieve the best results, we traversed through a series of data augmentations and model optimizations.

1.1 Objective

The objective of our research is to examine the shortcomings of an elementary CNN model and explore enhancements and extensions to improve categorization results. We will do so by measuring and comparing the training, validation, and test results of different CNN-based models.

2 Related Works

2.1 Duplicated Convolution Layers

Different variations of CNNs have been extensively applied in the research and development sectors of the signal analysis and music industries. For example, in their research paper, Yang and Zhang [1] used a Duplicated CNN to increase the predictive accuracy of their network in respective genres. Their network consisted of four parallel pooling layers, each with different types of layers (Max Pooling, Convolution, Average Pooling, or Dense layers).

Using the GTZAN dataset, they discovered that each pooling layer best predicted a specific genre, thus allowing the collective network to achieve higher predictive accuracy. Instead of losing data after many convolutions, the network will retain the information through other parallel convolution and pooling layers. Their results also indicated that due to diverse set architectures used for different pooling layers, some genres are inherently simpler to classify than others.

2.2 Recurrent Neural Networks

In Montreal, researchers Zaragoza and Pertusa [2] applied a modified CRNN to address the Optical Music Recognition (OMR) task. Inputting a set of monophonic images to their network, they successfully reproduced each musical score with the average classification error 2% on the symbol level. Their network consisted of a series of convolutional filters for feature extraction and recurrent block modules to cover the sequential nature of the music. The set of images consisted of 94,983 random sequences of western music with 50 different possible layers. Consequently, their work proved the potential of end-to-end architecture for OMR using the CRNN design and widened the possible applications for convolutional deep learning.

3 Methodologies

3.1 Architectures

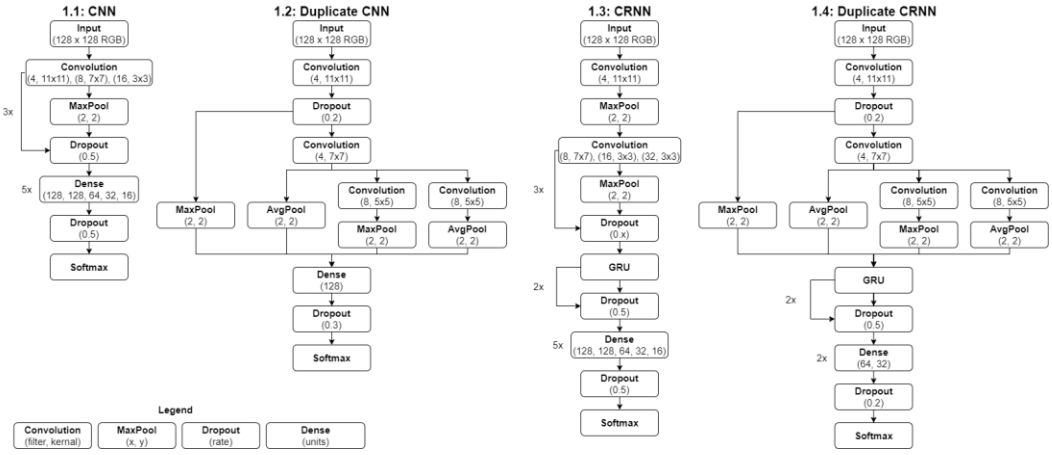

Figure 1: The diagrams above illustrate our base CNN model and the augmentations that were made to improve expected accuracy such as the CRNN, DCNN, DCRNN

All models take an image as input and use ReLU activation, as it is the most common in CNN models and produces the most consistent results during testing. Other general conventions utilized are increasing filters and decreasing kernel sizes in subsequent convolution layers, because initial input data is generally noisy, and useful lower-level features are difficult to extract, thus higher-level features are extracted first with larger convolution windows in order to capture larger patterns. At the end of each model, a dense layer with the softmax activation is used to classify our image. The models use cross entropy loss, since it is the most common loss function for multi-class classification problems, and the Adam optimization algorithm, which adapts the learning rate of our models during training.

3.1.1 Base CNN

The elementary CNN model, depicted in figure 1.1, will serve as our baseline model. The CNN is loosely based on the work of Mingwen Dong and is mostly composed of feature extracting convolution layers [3]. It performed with 37.75% overall accuracy during tests.

3.1.2 Duplicated CNN

The model in figure 1.2 integrates duplicated layers into the basic CNN. This model is partly inspired by the network used by Yang and Zhang [4], in which they use duplicated convolutional layers. This method allows for less data loss after each layer since data is being passed through parallel convolution and pooling layers separately, instead of through multiple subsequent instances of those layers. Each parallel route contains a varying number of convolution layers (to extract different levels of features) and distinct pooling layers to emphasize distinct features. Once all the parallel layers are completed, their outputs are concatenated into one output, and then flattened for the dense layers.

3.1.3 Sequential CRNN

Figure 1.3 illustrates the sequential CRNN model that contains the CNN outlined in 3.1.1 to extract various levels of features, an RNN module to modify the output vector of the CNN within the context of previous output vectors using GRUs, and an output layer to classify the spectrogram. At the end of the CNN module, batch normalization was implemented to stabilize the learning process. For the RNN module, we were required to reshape our input for the recurrent layers, which decides what data is important to keep, and what data can be "forgotten" based on previous data.

3.1.4 Duplicated CRNN

The last model integrates the duplicated CNN module from 3.1.2 into our sequential CRNN from 3.1.3, theoretically integrating the benefits of duplicate layers with the long-short term memory RNN.

3.2 Dataset and Features

To train and test our models, we used a small subset of FMA called FMA_small, which consists of 8 balanced genres each with 1,000 songs in MP3 format that are 30 seconds long for a total of 8,000 songs [5]. With FMA_small, we divided the dataset into 80% training, 10% validation and 10% testing splits to be used by our models

3.2.1 Dataset Preparation and Feature Extraction

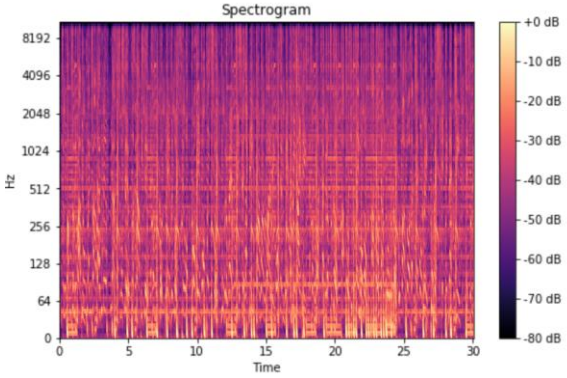

Figure 2: A spectrogram image of song from the hip-hop genre

Before the .mp3 files from the dataset can be used to train and test the model, we have to convert the files into mel-spectrogram images. Example of a mel-spectrogram image is shown in figure 2. To convert .mp3 to mel-spectrograms, we must first convert our .mp3 into .wav files to be able to utilize Librosa. Afterwards, we convert the .wav files into mel-spectrograms images with 16000Hz sampling rate.

3.2.2 Dataset Augmentation

Two augmentations were performed on the raw audio data in the training set (pitch-shifting and tempo change) which afterwards gets converted into mel-spectrograms. For pitch-shifting, the audio files were shifted 1 semitone up and 1 semitone down. For tempo changes, the speed of the audio files was slowed to 0.93x and sped up to 1.07x. As the training set consists of 6,400 audio files, every augmentation generates 6,400 more data files. As a result, the combined dataset of original data + augmented data totaled to 32,000 spectrogram images.

3.3 Batch Size

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\left(\frac{1}{|B|}\sum_{x\in B}\nabla f\left(x, \boldsymbol{w}_t\right)\right)$$

Figure 3: Equation for weight update using SGD optimizer. "B" represents the batch size, η is the learning rate, f(x, $\mathbf{w}_t$) is the function of our model and w_t is the weight

The batch size hyperparameter represents the number of training samples used in one update during training. By increasing the batch size, we can reach similar or better test accuracies in the same number of epochs with fewer parameter updates as shown in the papers by Pavlo M. Radiuk [6] and by Smith et al [7]. Radiuk concluded that the increase in batch size correlates with image recognition accuracy due to the better gradient averages as shown in the weight update equation for stochastic gradient descent (SGD) in figure 3 [7]. Thus, the increase in batch size will produce a better estimate of the gradient for the entire dataset, which results in a better weight update [6]. In addition, Smith et al also shows that a better weight update will result in the model achieving similar test accuracies as lower batch sizes in fewer weight updates [7]. In our experiments, we examine batch sizes 4, 16, 32, and 64 to show the improvements in training speed and classification accuracy of our models.

3.4 Optimizers

Another improvement to our models is the choice of optimizers. There are four optimizers in our experiments; Adadelta, Adam, RMSprop, and SGD. Both Adadelta and Adam are gradient-based optimizers that computes adaptive learning rates for each parameter, assigning smaller learning rates to frequently used parameters and larger learning rates to infrequent ones [8]-[9]. Where they differ is that Adam utilizes momentum to escape saddle points. RMSprop uses the sign of the gradient to perform updates and adapts the step-size for each parameter, while also keeping a moving average of the squared gradient for each weight to allow for mini-batches [10]. Lastly, SGD is an optimizer that performs stochastic gradient descent. To determine the best performing optimizers, we will train our models utilizing each optimizer and compare the results.

3.5 Cyclic Learning Rates

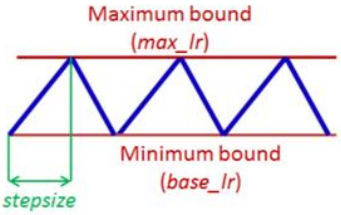

Figure 4: Illustration of the triangular CLR policy

Cyclic learning rates (CLR), created by Leslie N. Smith, is a method that alters the learning rate to cyclically vary between specified boundaries, instead of monotonically decreasing [11]. According to Smith, the difficulty in minimizing loss arises from saddle points as they have small gradients that slow the learning process [11]. By increasing the learning rate, we can combat small gradients

and traverse saddle point plateaus. In our experiments, we use the "Triangular" CLR policy with the parameters being step size of 8 epochs, base learning rate of 0.0005 and a max learning rate of 0.001. The triangular policy, as illustrated in figure 4 [11], linearly increases from base_lr to max_lr, then decreases linearly from max_lr to base_lr with respect to the stepsize.

4 Experiments and Results

4.1 Hyperparameter Tuning

4.1.1 Batch Size

Batch Size	4	16	32	64
CNN	35.49%	27.37%	35.37%	37.75%
DCNN	38.12%	35.00%	37.87%	36.50%
CRNN	26.37%	41.85%	41.25%	38.12%
DCRNN	42.50%	40.63%	44.12%	42.63%

Table 1: Test accuracy of our models for various batch sizes

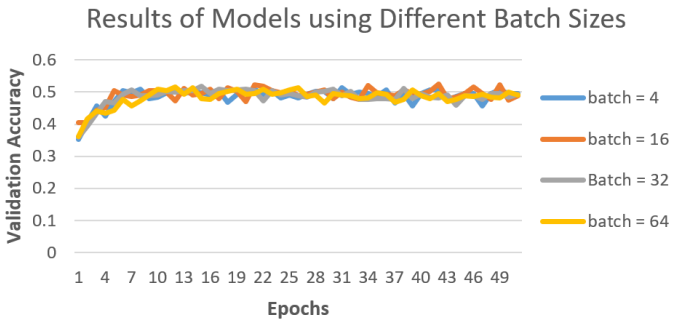

Figure 5: The validation accuracy of DCRNN with various batch sizes

From our results in the table in table 1, we see that our models did not receive the image recognition accuracy improvement as proposed in Radiuk's paper [6]. We see that lower batch sizes for DCNN and CRNN models were able to achieve a higher test accuracy for batch size 4 than when using batch size 64. However, for CNN and DCRNN, we were able to obtain similar test accuracy for batch size 4 and 64, which Radiuk also stated could happen. Radiuk did mention that the optimal value for batch size to receive accuracy improvement is for batch sizes 200 and up, which could be the reason we don't receive any substantial improvements as we only use batch sizes up to 64. A reason for lower batch sizes performing better than higher batch sizes, could be for lower values the average gradient used when updating the model would be nosier, which can help the model escape a local minimum and explore other minimas. From the graph in figure 5, we don't see much of a difference in training speed either, as proposed by Smith et al's, which again could be due to our range of batch sizes [7].

4.1.2 Cyclic Learning Rates

	Standard Learning Rate		Cyclic Learning Rate	
	Epochs Required	Test Accuracy	Epochs Required	Test Accuracy
CNN	29	37.75%	15	36.63%
DCNN	9	36.50%	7	41.63%
CRNN	47	38.13%	48	43.38%
DCRNN	12	42.63%	23	40.00%

Table 2: Test accuracy of our models using non-CLR and CLR with the epochs required

From table 2, we observe gains in test accuracy and faster training convergence that were mentioned in Smith's paper [11]. Also, every model improved from utilizing CLRs except for DCRNN by achieving similar or better accuracy with fewer epochs. One possible reason its poor results is due to the base and max learning that were chosen to be cycled over.

4.1.3 Optimizers

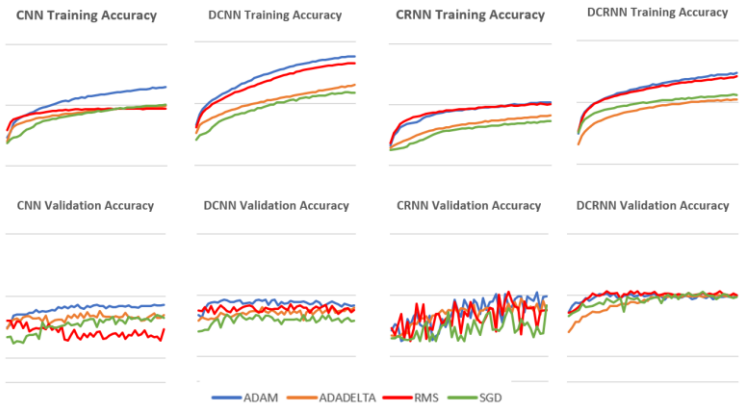

Figure 6: Graphs illustrating training accuracy (0.0 – 1.0) vs. epochs (0-49)

During training, the ADAM optimizer (blue) had the best overall performance across all models as observed in the graph above, with RMS closely behind. Adam is observed to perform the best in validation, with less oscillation and higher validation accuracy. This is noticeable especially in the CNN models as shown in figure 6. This is expected, as it performed significantly better than other optimizers during training in figure 6.

4.2 Results of Final Models

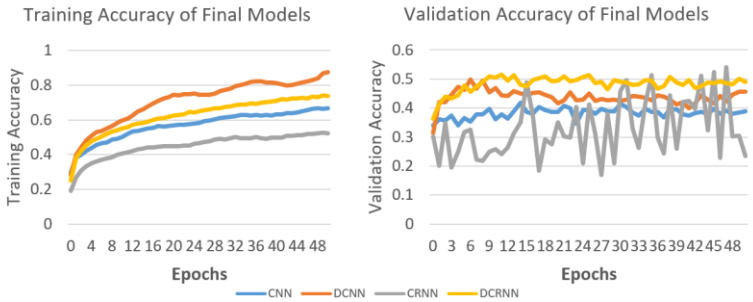

Figure 7: Training and validation accuracy of our final tuned models

Through our hyperparameter experiments, we found that Adam was a consistent and effective optimizer for all our models and CLR provided great improvements to all our models except DCRNN. After applying the hyperparameter tuning and running our model with batch size 64, in order to speed up training, we arrive at the results in figure 7 and table 3. From the table we see that each of our improved models performs better for testing accuracy over our standard CNN. DCNN, was able to train faster in half the epochs and achieve higher test accuracies. CRNN, achieved substantially higher validation and test accuracy, but required many more epochs, which is due to the large number of parameters required to be trained. Standard CNN had ~300,000 parameters where CRNN base had ~1,000,000 parameters. Also, CRNN, from the graph we see that the validation was sporadic for its validation accuracies, which could be a result of a learning that was too large. DCRNN combines the best of the two outcomes of DCNN and CRNN, it was able to reach high test accuracies over standard CNN and in less epochs. We also see in the validation accuracy graph, that DCRNN had a more stable curve.

	Epoch Required	Validation Accuracy	Test Accuracy
CNN	15	41.75%	36.63%
DCNN	7	49.75%	41.63%
CRNN	48	54.00%	43.38%
DCRNN	12	51.50%	42.63%

Table 3: Validation/test accuracy of our models after hyperparameter tuning

Overall, there were many observable improvements produced by the various CNN augmentations, especially in the CRNN models. As shown in the bar graph (figure 8), every genre excluding international and experimental had significant accuracy improvements across all models. From this we can infer that they seem to be able to detect beat-heavy samples such as hip-hop with ease, but struggles with high variance, and highly detailed genres such as instrumental and experimental.

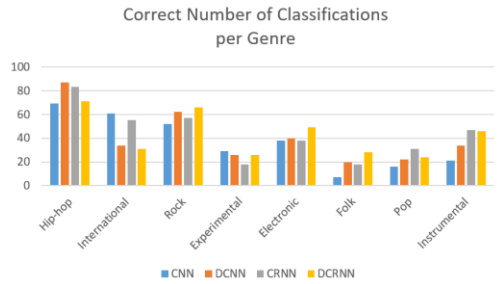

Figure 8: Percentage of correct classifications per genre for each of our tuned models

5 Conclusion

5.1 Problems

Our dataset only consists of 1000 samples per genre, which does not include enough variation to confidently generalize the genres. However, the larger datasets (FMA_medium or FMA_large) are designed with more genres, making it harder to classify as more genres can overlap. In addition, they are not balanced, thus potentially leading to a skewed accuracy towards genres with more samples.

Another issue is that some songs in the dataset are vague, with poor representation of the genre. An example is “No Good Deed Goes Unpunished” by “Eva Aguila”, which is simply loud static, which produces a mel-spectrogram that is completely yellow. An image with extreme intensity in dB in all areas of the image is a poor representation of the experimental genre and causes weight updates to be skewed. The problem of poor representation extends to genres with high variance such as experimental, international, and instrumental. The latter two can include other genres, for example, the spectrogram of a drum instrumental contains many similarities to rock, or an international song can also be considered another genre in a different language. In addition, due to hardware constraints, images were set to a resolution of 128x128, which affects fidelity and detail in songs that may be the differentiating factor between genres.

5.2 Conclusion and Future Works

In conclusion, our experiments demonstrate overall improved test accuracy with faster learning with the addition of RNN and duplicate convolution layers to the basic CNN model. Improvements in results could also be realized with the removal of genres with poor representation, utilizing raw data in addition to the spectrogram, and by increasing hardware capacity to boost parameter count and image resolution. This would only further prove our studies and the applicability of these model architectures.

Contributions

Link to Github Repository: https://github.com/AndyLiuCodes/Music-Genre-Classification-with-CNN-Hybrid-Architectures

Contributions made by **Henry Yip** includes the model design for duplicated CNN and duplicated CRNN, along with the writeup of section 3.1 Architecture, 4.1.3 Optimizers, parts of 4.2 Results of Final Models, 5.2 Conclusions and Future Works, and the diagrams pertaining to aforementioned sections.

Contributions made by **Avital Vetshchaizer** include: Writing the paper's Abstract, section 1 (Introduction), section 2 (Related Works), parts of section 3 (Methodologies), and parts of section 5 (Conclusion). I also designed and completed the poster.

Contributions made by **Andy Liu** include dataset processing, coding CNN and CRNN models, data collection of all models. Also, contributed to the written sections that include problems of dataset (5.1), results (4.2), CLR(4.1.2), batch size (4.1.1), CLR (3.5), optimizers (3.4), batch size (3.3), dataset augmentation (3.2.2), dataset (3.2), and dataset preparation (3.2.1).

References

[1] H.Yang & W.Q. Zhang (2019) Music Genre Classification Using Duplicated Convolutional Layers in Neural Networks, ISCA

[2] J. Calvo-Zaragoza, J.J. Valero-Mas & A. Pertusa (2017) End-to-End Optical Music Recognition Using Neural Networks, ISMIR, 2017

[3] M. Dong (2018) Convolutional Neural Network Archives Human-Level Accuracy in Music Genre Classification, Cornell University ArXiv

[4] H.Yang & W.Q. Zhang (2019) *Music Genre Classification Using Duplicated Convolutional Layers in Neural Networks,* ISCA

[5] M. Defferrart, K. Benzi, P. Vandergheynst & X. Bresson (2017) FMA: A Dataset For Music Analysis, Cornell University ArXiv

[6] P.M. Radiuk (2017) Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets, Information Technology and Management Science

[7] S.L. Smith, P. Kindermans, C. Ying & Q. Le (2018) Don't Decay The Learning Rate, Increase The Batch Size, OverReview.org

[8] M.D. Zeiler (2012) Adadelta: An Adaptive Learning Rate Method, Cornell University Arxiv,

[9] D.P. Kingma & J.L. Ba (2015) Adam: A Method For Stochastic Optimization, Cornell University ArXiv

[10] G. Hinton, N. Srivastava & Kevin Swesky (2014) Overview of Mini-Batch Gradient Descent, University of Toronto

[11] L.N. Smith (2017) Cyclical Learning Rates for Training Neural Networks, Cornell University ArXiv